第8章

スペクトラム・アナライザの製作

――ディジタル信号処理にFPGA内蔵の乗算器を活用する

内藤竜治

本誌付属のFPGA基板のディジタル信号処理への活用例を紹介する．ここでは，音声帯域のスペクトラム・アナライザを製作する．Spartan-3が内蔵する乗算器を活用してFFT（fast Fourier transform）を行う．JTAG（Joint Test Action Group）インターフェースによる通信機能を活用し，表示・制御にパソコンを利用している．（編集部）

Spartan-3には高性能なハードウェア乗算器が内蔵されています．付属FPGA基板に搭載されているXC3S50には四つの乗算器が内蔵されており，これを使って何かおもしろいディジタル信号処理ができないかなと考え，高速フーリエ変換（FFT：fast Fourier transform）を用いた音声帯域スペクトラム・アナライザを作りました（**写真1**）．本機を「フーリエ次郎」と名づけます．

1. 基本設計

まず第一に，付属FPGA基板をだれにでも気軽に楽しんでもらえるように，外付け回路がなくても動作することを前提としました．つまり，付属FPGA基板に，電源と水晶発振器とJTAG（Joint Test Action Group）ケーブルをつないだだけの最低限の構成で動作が確認できるようにします．操作はホスト・パソコンから行いますが，ホスト・パソコンとの接続をできるだけ簡便にするため，通信にはFPGAの書き込みにも使われるJTAG信号を使います．めんどうな配線は必要ありません．

また，次のステップとして，A-DコンバータとOPアンプ，そして多少の抵抗とコンデンサをつなぐことにより，音声帯域のスペクトラム・アナライザを構成することにしました．

● 二つの動作モードに対応

本機は，FFT計算モードとスペクトラム・アナライザ・モードの二つの動作モードがあります．

FFT計算モードは，ホスト・パソコンから送ったデータをFFT演算してホスト・パソコンに送り返すモードです．これは最低限の構成のボードで動作します．

スペクトラム・アナライザ・モードでは，マイクで拾った音声をA-D変換し，FFTしてホスト・パソコンに送ります．これは，いわゆる音声帯域用スペクトラム・アナライザとして動作します．

8

● コンデンサ・マイク，ADC，OPアンプなどで構成

図1に回路図を示します．コンデンサ・マイクとA-Dコンバータの間にOPアンプ（米国National Semiconductor

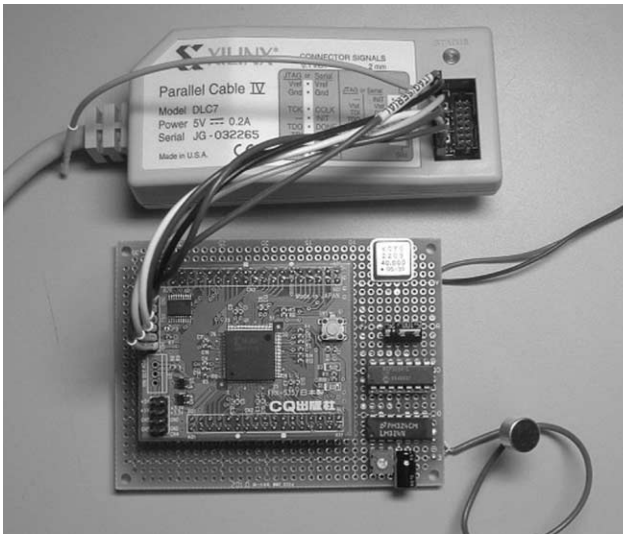


写真1　音声帯域スペクトラム・アナライザの外観
JTAGインターフェースを使い，パソコンで制御・表示する．

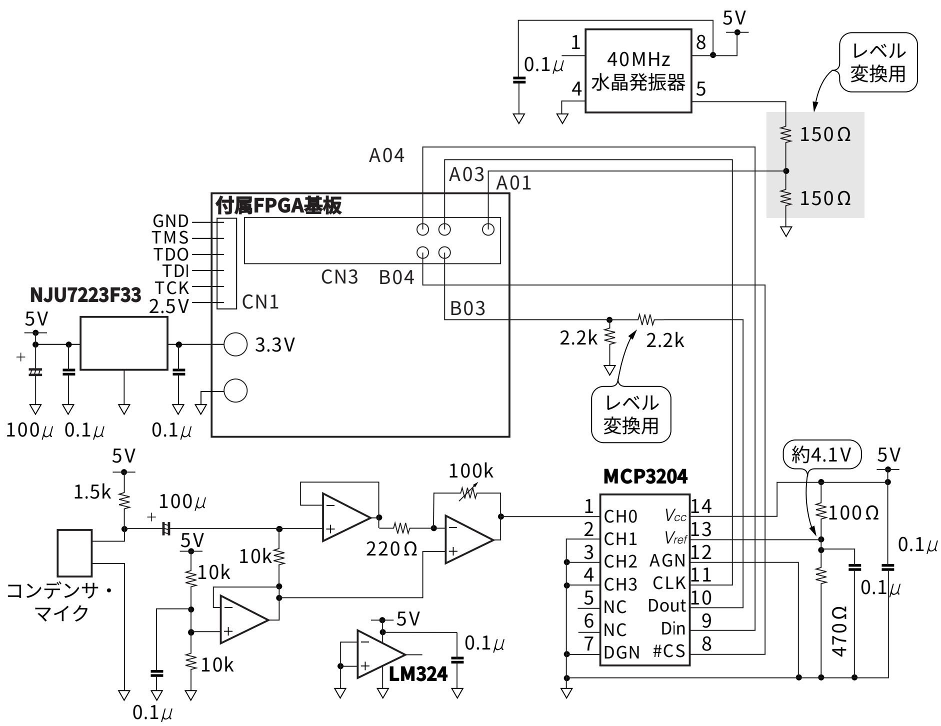


図1　音声帯域スペクトラム・アナライザの回路図

表1　設計したFPGA機能の概要

項　目	内　容
データ数	512ポイント
精度	16ビット固定 小数点
使用Block RAM	3個
使用乗算器	3個
FFT速度	691.2μs
ホストとの通信 方法	USER1命令に よるJTAG通信
スライス使用率	44% （768スライス中 341スライス）
使用I/O数	7本
最大動作速度	136.129MHz
そのほかの 特記事項	任意形状の 窓関数を処理 可能

社のLM324）を入れ，ゲインを調整できるようにしています．A-Dコンバータは米国Microchip Technology社のMCP3204（12ビット，100kHz）を使います．MCP3204は，0V～約4Vの電圧を，100kHzのサンプリング周波数で12ビットの2進整数に変換します．

写真1のように，本機はユニバーサル基板を使って製作しました．FPGAボードのほかに，3.3Vレギュレータ，OPアンプ，A-Dコンバータ，電解コンデンサなどを実装しています．裏面にはいくつかのチップ部品が載っています．

● FPGAに16ビット固定小数点FFTを実装

本機の中心となるFPGAの仕様を**表1**に示します．

FPGAでは，512ポイントの16ビット固定小数点のFFTを行います．FFTではバタフライ演算という計算を繰り返し実行します．512ポイントの場合は，全部で2,304回のバタフライ演算を行うことになります．

この設計では，1回のバタフライ演算を12クロックで実行するので，1回のFFTには27,648クロックを要します．クロックが40MHzの場合は1回のFFTを約690μs，すなわちFFTだけならば毎秒1,447回計算できることになります．

なお，詳しい理由は後述しますが，データ量を512ポイントとしたのはメモリ容量との兼ね合いによるものです．

FPGAのアーキテクチャとデータ・フローを**図2**に示します．バタフライ演算器とメモリ・ブロックを中心として，全体を制御するステート・マシン，FFT時にメモリなどを管理するコントローラ，ホスト・パソコンとの通信を行うJTAG通信回路，A-Dコンバータからのサンプリング・データを格納・制御するユニットなどから構成されています．

● パソコンで動作する制御ソフトウェアを作成

パソコンで動作する制御ソフトウェアの画面を**図3**に示します．Windows 98/2000/XPに対応しています．

FFT計算モードでは，パソコン上で作成したデータを本機に送ってFFTすることができ，サンプルの関数をFFTした結果を楽しむことができます．

スペクトラム・アナライザ・モードでは，FFT計算モードの機能のほか，マイクで拾った音声をリアルタイムにスペクトラム表示することができます．

2. FFT演算のくふう

データの取り扱いの利便性から，数値の精度は16ビット

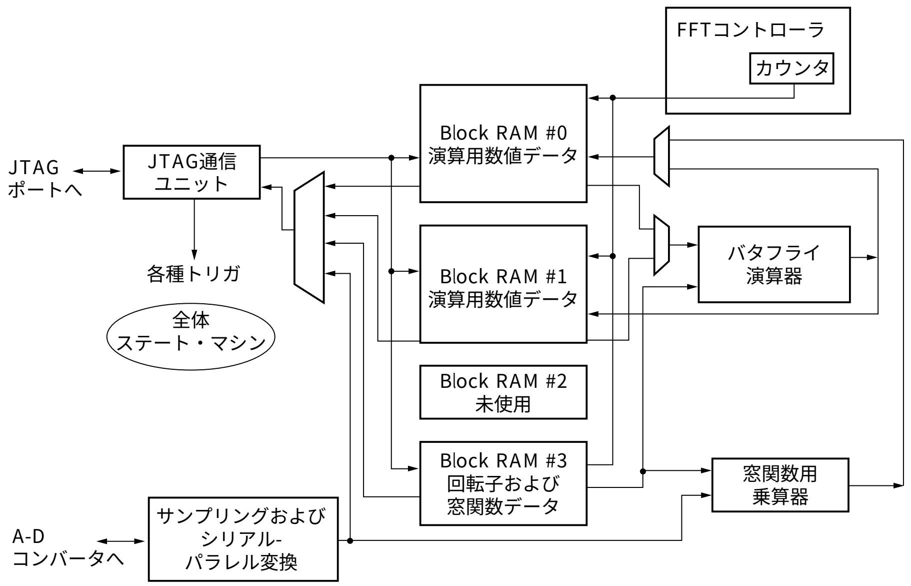


図2　FPGAのアーキテクチャとデータ・フロー

バタフライ演算器とメモリ・ブロックが機能の中心である．ほかに，全体を制御するステート・マシン，FFT時にメモリなどを管理するコントローラ，ホスト・パソコンとの通信を行うJTAG通信回路，A-Dコンバータからのサンプリング・データを格納・制御するユニットなどがある．

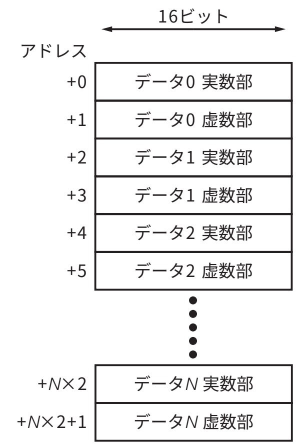


図4　複素数データの並び

読み出すメモリのアドレスをインクリメントするたびに，実数，虚数，実数，虚数…とデータが出てくる．

固定小数点とします．ただし，扱う数値の範囲は絶対値が1未満であるとします．すなわち，0x0000を0とし，0x7fffを正の数の最大値(0.9999695)，0x8001を負の数の最小値(−0.9999695)とします．±1は使用しません．詳細は後述しますが，これは計算の過程で絶対値が1を超えないようにするための配慮です．

フーリエ変換前の時系列データも，フーリエ変換後のスペクトルも，各点の値は−1〜1の数値になります．

● 複素数の掛け算を行う

二つの複素数の掛け算は，

$$(a+bi)\times(c+di)=(ac-bd)+(bc+ad)i$$

a，b，c，dは実数，iは虚数単位

となります．本機で扱う数値の絶対値は1未満なので，二つの複素数の掛け算を行っても絶対値が1を超えることはありません．$ab-bd$と$bc+ad$には加減算が含まれていますが，上記の理由によってオーバフローすることはありません．

● 複素数データの並びを規定

本回路で使用する演算用データと回転子(ひねり係数)は，

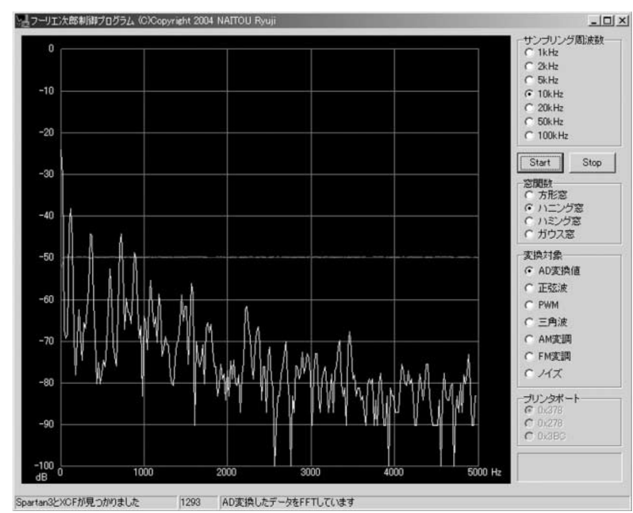


図3　制御ソフトウェアの画面

スペクトラム・アナライザ・モードのようす．

図4のような順序でメモリ上に並んでいます．読み出すメモリのアドレスをインクリメントするたびに，実数，虚数，実数，虚数…とデータが出てきます．これは次段のバタフライ演算器に入れるのにつごうが良くなっています．

● バタフライ演算とは

バタフライ演算は，式(1)のように三つの複素数A，B，W間で行う演算で，二つの結果J，Kが出力されます．

$$J=A\times W+B$$
$$K=A\times W-B \qquad (1)$$

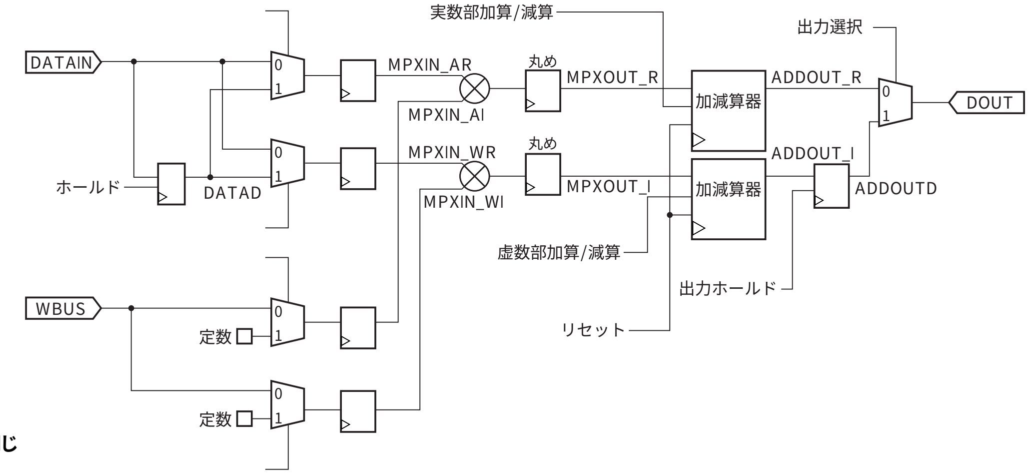


図5
バタフライ演算器の構造
入力される実数と虚数のデータは，同じバスを時分割で共有している．

$A \times W$とBの絶対値はそれぞれ1未満ですが，加算と減算の結果は1を超えてしまうことがあります．そこで，乗算と同時に値を1/2にすることで，バタフライ演算の結果の絶対値がかならず1未満に収まるようにしています．

これでは一見，精度が失われるかのように思えますが，実は512ポイントのFFTで求めるスペクトルは，最終的には配列全体を1/512しなければなりません．FFT前に時系列に並んでいる各点のデータは，FFTの過程で計9回のバタフライ演算を通るので，結果としてちょうどぴったりなのです．

● 乗算器と加減算器でバタフライ演算器を構成

複素数の乗算は，一般には4回の実数の乗算で行われます．今回設計するバタフライ演算器では，複素数の実数部分と虚数部分をそれぞれ16ビットのレジスタに格納して，同時に計算できるようにします．よって，本回路は2個の乗算器を使用します．また，二つの乗算器の後ろには二つの加減算器があり，乗算の結果を加減算して積算することができます．

設計したバタフライ演算器の構造を**図5**に示します．データ・フローは**図6**のとおりです．入力される実数と虚数のデータは同じバスを時分割で共有しており，演算途中でうまく処理されるしくみになっています．

バタフライ演算器では式(1)のJとKを求めますが，式(1)を一見しただけでわかるように，

$$K = J - 2 \times B$$

です．よって，本回路ではいったんJを求めた後，Bを2回減じることでKの値を求めています．

● FFTアルゴリズムを改良

よく目にするJ. W. CooleyとJ. W. TukeyのFFTアルゴリズムは，**図7**のようなデータの流れになります．これをソフトウェアで記述すればすぐにわかりますが，この演算を普通に行うと3重のfor文のループになってしまいます．しかも，内側の二つのループは単純なfor文ループではなく，ループ回数が毎回変わるめんどうなものになります．

この3重のループをそのまま回路で実現しようとすると非常にやっかいです．そこで，バタフライ演算の計算順序とメモリへの格納方法を入れ替えることで，3重のループを1重のループにほぐすことを考えました．改良したFFTのアルゴリズムは，**図8**のようなデータの流れになります．

このアルゴリズムでは，バタフライ演算の順序を変更しているだけなので，演算回数は通常のFFTと変わらず，$N \times \log N/2$のままです．しかし，ループが1重になったことと，次に計算するべきアドレスや格納すべきアドレスが簡単に求まることにより，ハードウェアとして実装しやすくなります．

このアルゴリズムの動作を**図8**のような8ポイントFFTの場合を例に考えてみると，アドレス0番と1番のデータを演算してその結果を0番と4番に格納します．次に2番と3番のデータを演算してその結果を1番と5番に格納します．このように，アドレス・カウンタのビット・シフトと最上位ビットを‘1’にするか‘0’にするかという非常に単純な規則で，取り出すメモリのアドレスと書き込むメモリのアドレスが求められます．

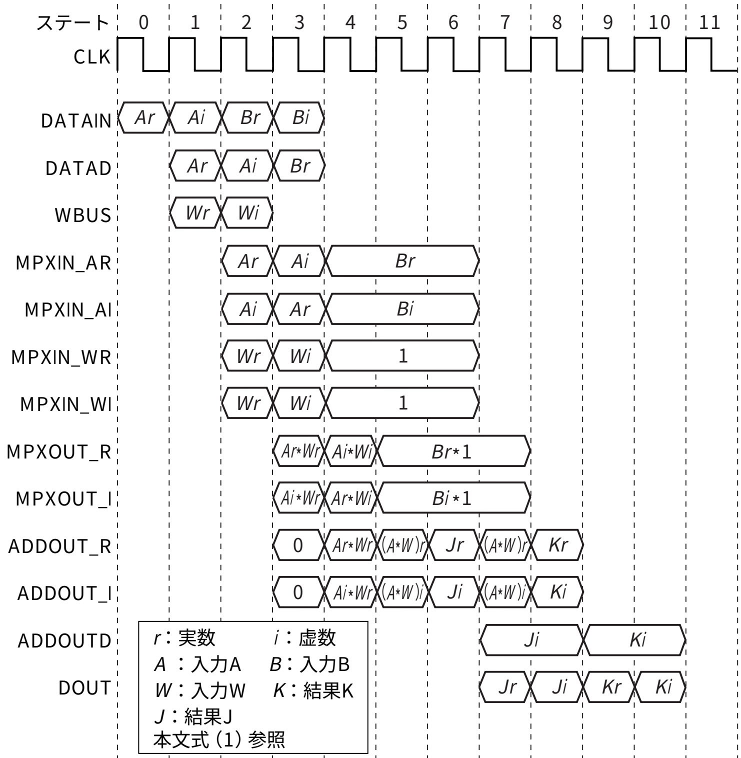


図6　バタフライ演算器のデータ・フロー
入力される実数と虚数のデータが演算途中でうまく処理される．

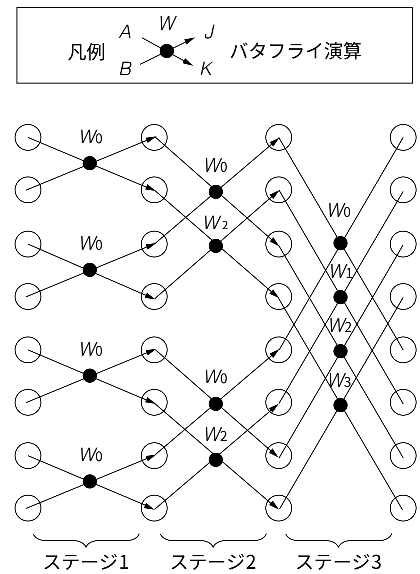


図7　通常のFFTのデータ・フロー
8ポイントFFTの場合．

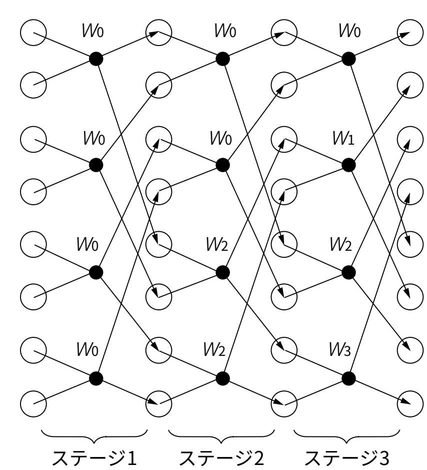


図8　設計したFFTのデータ・フロー
8ポイントFFTの場合．

また，回転子の並ぶ順序もすっきりとしています．例えば，通常の16ポイントのFFTの場合を考えてみると，回転子の添え字は{ 0, 0, 0, 0, 0, 0, 0, 0 },{ 0, 4, 0, 4, 0, 4, 0, 4 },{ 0, 2, 4, 6, 0, 2, 4, 6 },{ 0, 1, 2, 3, 4, 5, 6, 7 }となり，規則性を求めるのはちょっとめんどうです．

一方，改良されたアルゴリズムでは{ 0, 0, 0, 0, 0, 0, 0, 0 },{ 0, 0, 0, 0, 4, 4, 4, 4 },{ 0, 0, 2, 2, 4, 4, 6, 6 },{ 0, 1, 2, 3, 4, 5, 6, 7 }という順序になります．つまり，アドレス・カウンタの値をビット・マスクするだけで求めることができ，ハードウェアによる実装がとても簡単になります．

ただし，このアルゴリズムには一つだけ欠点があります．通常のFFTでは，0番地のデータと1番地のデータを演算した結果は，0番地と1番地に書き戻します．ところが，本アルゴリズムでは読み出したアドレスと書き込むアドレスは一致しないので，2バンク以上のメモリをインタリーブして使うことになります．また，2倍のメモリ容量が必要となります．

3. FPGA内蔵機能の活用

● BSCANを使ってJTAG通信

Spartan-3には，JTAGの信号を使ってFPGA内部のユーザ回路と通信できるようにするため，BSCANコンポーネントが用意されています．USER1とUSER2というJTAG命令を使い，TDI，TDO信号をユーザの用意したデータ・レジスタに接続できるようになります．

BSCANの使用例を**リスト1**に示します．なお，この例では多少簡略化しており，実際の本機の構成はもう少し複

雑です．

BSCANには，USER1とUSER2の命令に対応して二つの系統があります．BSCANを使うためには，任意の長さのシフト・レジスタを用意し，シフト・レジスタの入力にTDIを，シフト・レジスタの最下位ビットにTDOをつなぎます．

USER1を使う場合，シフト・レジスタはDRCK1でシフトされるようにします．FPGAからパソコンに送信するデータは，JTAG_CAPTUREがアサートされているときのDRCKの立ち上がりでシフト・レジスタに代入します．また，UPDATE信号はシフト動作が完了してデータが確定したことを示す信号です．UPDATEのときにはDRCKは出ないのでご注意ください．

また，JTAGのクロックとシステム・クロックは同期していないことに注意してください．**リスト1**の例ではUPDATE信号をシステム・クロックで同期化していますが，本来はUPDATE信号をクロック入力として使ってシフト・レジスタから内部のデータ・レジスタへデータを移すのが正しいのでしょう．さらに，CAPTURE信号もシフト・レジスタの非同期セットに用いるのが正しいのかもし

リスト1　BSCANの使用例

```
signal CLK              : std_logic;--システム・クロック
signal SHIFTREG         : std_logic_vector(23 downto 0); --シフト・レジスタ
signal COMMAND_TO_FPGA  : std_logic_vector(23 downto 0); --FPGAに与えるコマンド
signal STATUS_OF_FPGA   : std_logic_vector(23 downto 0); --ホストへ送るステータス
signal UPDATE           : std_logic; -- BSCANコンポーネントのUPDATE信号
signal UPDATED          : std_logic; -- UPDATE信号をCLKでサンプリングしたもの
signal UPDATED2         : std_logic; -- UPDATED信号をCLKで遅延させたもの

component BSCAN_SPARTAN3
     port (
              CAPTURE : out STD_ULOGIC;
              DRCK1   : out STD_ULOGIC;
              DRCK2   : out STD_ULOGIC;
              RESET   : out STD_ULOGIC;
              SEL1    : out STD_ULOGIC;
              SEL2    : out STD_ULOGIC;
              SHIFT   : out STD_ULOGIC;
              TDI     : out STD_ULOGIC;
              UPDATE  : out STD_ULOGIC;
              TDO1    : in STD_ULOGIC;
              TDO2    : in STD_ULOGIC
     );
end component;

          . . .
                                                       ↗
```

```
↘ process(CLK) begin
       if(DRCK1'event and (DRCK1='1')) then
              --シフト動作はDRCKで行う
              if((SEL1 = '1') and (CAPTURE = '1')) then
              --FPGA内のステータス情報をシフト・レジスタに代入
                     SHIFTREG <= STATUS_OF_FPGA;
              elsif(SEL1 = '1') then
              --SEL1がアサートされていればシフト
                     SHIFTREG <= TDI & SHIFTREG(23 downto 1);
              end if;
       end if;

       if(CLK'event and (CLK='1')) then
              UPDATED  <= UPDATE and SEL1;
              UPDATED2 <= UPDATED;
              --システム・クロックで同期化する

              if((UPDATED2 = '0') and (UPDATED = '1')) then
                     COMMAND_TO_FPGA <= SHIFTREG;
              end if;
       end if;
  end process;
  TDO1 <= SHIFTREG(0);
```

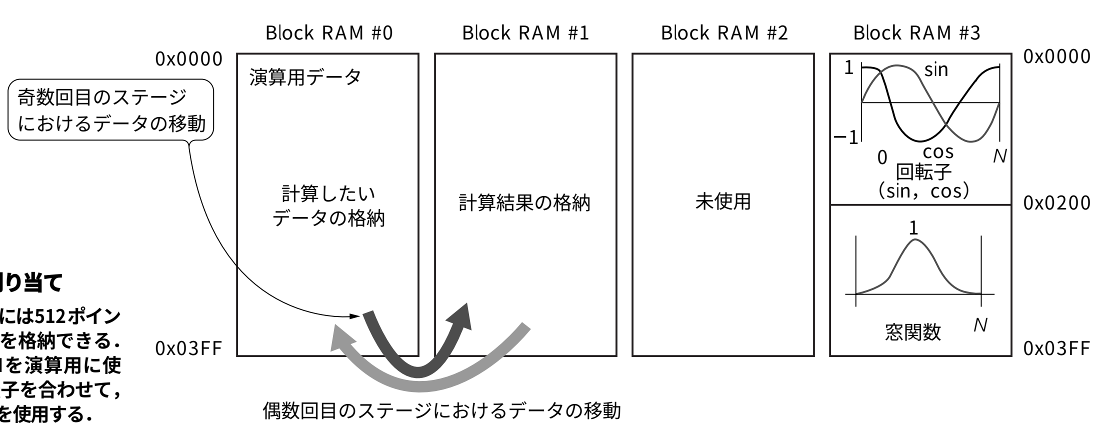


図9
Block RAMの割り当て
一つのBlock RAMには512ポイントの複素数データを格納できる．二つのBlock RAMを演算用に使う．窓関数と回転子を合わせて，一つのBlock RAMを使用する．

れません．

JTAGのしくみや，JTAGのデータ・レジスタとは何かということについては，参考文献(5)を参照してください．なお，BSCANの概要については，ISE WebPACKがインストールされているディレクトリの中の\doc\japanese\de\libsにbscan_spartan3.pdfというファイルがあります．

● 三つのメモリ・ブロック(Block RAM)を利用

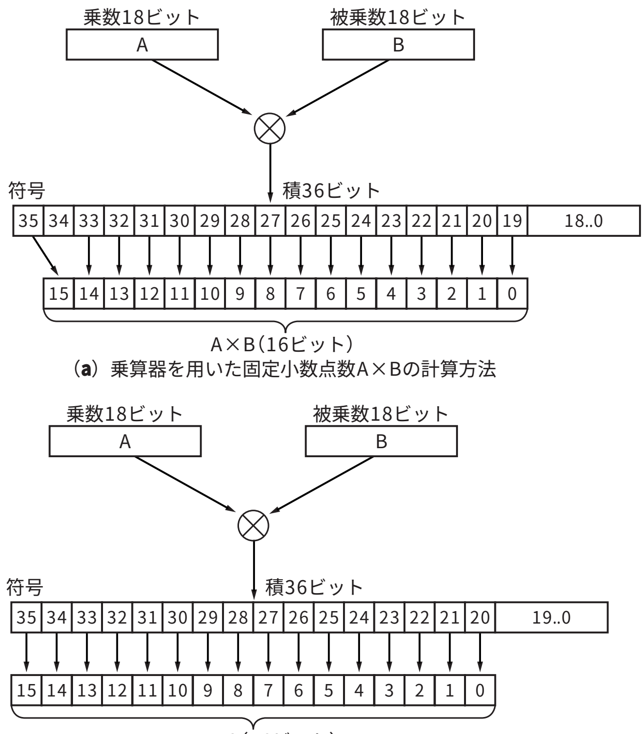
XC3S50には四つのメモリ・ブロック(Block RAM)が内蔵されています．一つのサイズは18Kビットなので，18ビット幅で1Kワードのメモリとして使用することができます．

本設計で扱うデータは16ビット幅です．一つのBlock RAMには1,024ワードのデータを入れることができますが，複素数で計算するので一つのデータにつき2ワード使います．よって，一つのBlock RAMには512ポイントの複素数データが格納できることになります．本機では，改良されたFFTアルゴリズムを用いるため，Block RAMは2バンク必要になります．したがって，二つのBlock RAMを演算用に使います．

回転子はW_0，…，W_{255}の256個の複素数なので，全部で512ワードを使います．窓関数は512ポイント分ありますが，こちらは実数値なので512ワードですみます．窓関数と回転子を合わせて16ビット1,024ワードになるので，バンク3のBlock RAMにぴったり納まります．

以上をまとめると，Block RAMの割り当ては**図9**のようになります．

また，Block RAMは真のデュアルポート・メモリです．本設計ではポートAをBSCANコンポーネントから操作されるポートとし，ポートBをFFTの演算で用います．この



(**a**) 乗算器を用いた固定小数点数A×Bの計算方法

(**b**) 乗算器を用いた固定小数点数A×B÷2の計算方法

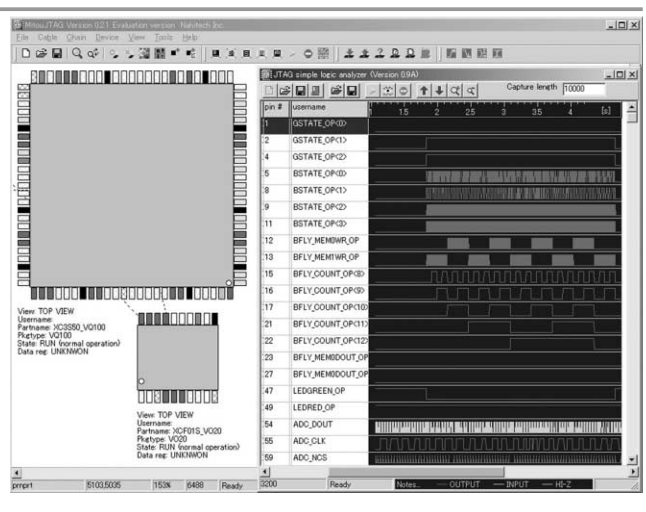
図10　固定小数点の乗算
18ビットどうしの乗算結果から16ビットの演算結果を得る．

コラム　FPGAのデバッグ法

オシロスコープやロジック・アナライザがないとFPGA回路のデバッグはとてもたいへんです．JTAGケーブルでFPGAの端子の状態を調べたり，ロジック・アナライザのように波形を観測できると便利です．ケーブルをつなぎかえることなく，FPGAやコンフィグレーションROMへの書き込みも行いたいものです．

筆者は，これらの機能を持つFPGA開発ツールを開発しています．評価版は下記のアドレスからダウンロードできます．

http://www.nahitech.com/jtag/



図A
MITOUJTAGの画面
FPGAの余ったピンに内部信号を出してデバッグしているようす．

ように分けることで，データ・バスやアドレス・バスにマルチプレクサやアービタを用いることなく，両方から独立にアクセスすることができます．

Block RAM2は本機では使用しません．機能の拡張にご利用ください．

● 18ビットの乗算器で16ビット乗算を実行

Spartan-3に内蔵されている乗算器は18ビット符号付きの乗算を行うことができ，入力されたデータはつねに符号付き整数として解釈されます．

16ビットの符号付き整数の乗算を行うためには，下位2ビットに何らかのデータを詰めなければなりませんが，本回路では最下位ビットをコピーして詰めています．

18ビットどうしの乗算結果は36ビットになりますが，本回路で欲しいのは16ビットの演算結果なので，下位の何ビットかは切り捨てることになります．

本回路で使う固定小数点のデータ形式では，**図10(a)**のようにすることで乗算ができるようです．また，乗算の結果を2で割った数がほしいときには**図10(b)**のようにするとうまくいくようです．

これをVHDLで表すと**リスト2**のようになります．

なお，実際の設計では，バタフライ演算器内で**リスト3**のように，切り捨てられてる部分の最上位ビットを見て簡単な丸め処理を行っています．

4．ホスト・パソコンとの通信

ホスト・パソコンと本機の通信を行うために，24ビットのコマンド・レジスタとステータス・レジスタを用意しています．コマンド・レジスタに値を読み書きすることで，メモリへのアクセスやFFT開始などのコマンドを与えることができます．また，ステータス・レジスタの値を読み込むことで，メモリの内容やFFTの実行状況を知ることができます．なお，コマンド・レジスタとステータス・レジスタはJTAG経由でシリアルにアクセスされるので，コマンド書き込みとステータス読み出しは同時に行われます．

● コマンド・レジスタとステータス・レジスタ

コマンドの形式を**図11**に，ステータスの形式を**図12**に示します．

図13は，JTAG経由のメモリ・アクセス回路の原理的な構造です．FPGA内にアドレス・カウンタがあり，メモリの読み書きはこのカウンタの示すアドレスに対して行われます．カウンタはメモリの読み書きによって自動的にインクリメントされるほか，コマンドのビット20を‘1’にすると値をセットできます．

読み出されたデータのビット15～ビット0は，メモリの値か，もしくはA-Dコンバータから得られた現在のアナログ値(0～4095)です．A-Dコンバータの値を読み込むには，あらかじめビット22を‘1’にしたデータをコマンド・レジスタにセットしておきます．ビット22が‘0’の場合はメモリの内容が読み出されます．

リスト2　18ビットどうしの乗算結果から16ビットの演算結果を得るVHDLコード

```
signal DATA_A : std_logic_vector(15 downto 0);
signal DATA_B : std_logic_vector(15 downto 0);
signal DATA_P : std_logic_vector(15 downto 0); -- A*B
signal DATA_P2: std_logic_vector(15 downto 0); -- A*B/2
component MULT18X18
port (    A: in STD_LOGIC_VECTOR (17 downto 0);
          B: in STD_LOGIC_VECTOR (17 downto 0);
          P: out STD_LOGIC_VECTOR (35 downto 0)
          );
end component;

      . . .

U1 : MULT18X18 port map
      (         A => A,
                B => B,
                P => P
      );
A       <= DATA_A & DATA_A(0) & DATA_A(0);
B       <= DATA_B & DATA_B(0) & DATA_B(0);
DATA_P  <= P(35) & P(33 downto 19);              -- A*Bの値
DATA_P2 <= P(35 downto 20);                      -- A*B/2の値
```

リスト3 丸め処理

```
     -- A*B/2を求める
if(MPXOUT_R_P(19) = ‘1’) then --乗算結果の19ビット目を見る
     --繰り上がり
       MPXOUT_R <= MPXOUT_R_P(35 downto 20) + “0000000000000001”;
else
     --繰り上がりなし
       MPXOUT_R <= MPXOUT_R_P(35 downto 20);
end if;
```

参考までに，Block RAMの内容を読み出すコードを**リスト4**に示します．

FFTを実行するには，コマンド・レジスタのビット23を‘1’にします．この場合，FFTの実行には二つのモードがあります．一つはホスト・パソコンから時系列データをBlock RAMにあらかじめ書き込んでおく方法，もう一つはA-Dコンバータでサンプリングされたデータを FFTする方法です．後者のサンプリング・データの場合は，ビット21とビット23を‘1’にしてコマンドを与えます．ビット23を‘1’にしたときのビット6〜ビット0はサンプリング周期となります．この値をTとすると，サンプリング周波数は100kHz/（T+1）となります．

なお，読み出したステータスのビット23が‘1’の場合は，現在，FFTの処理を行っている最中であることを示します．

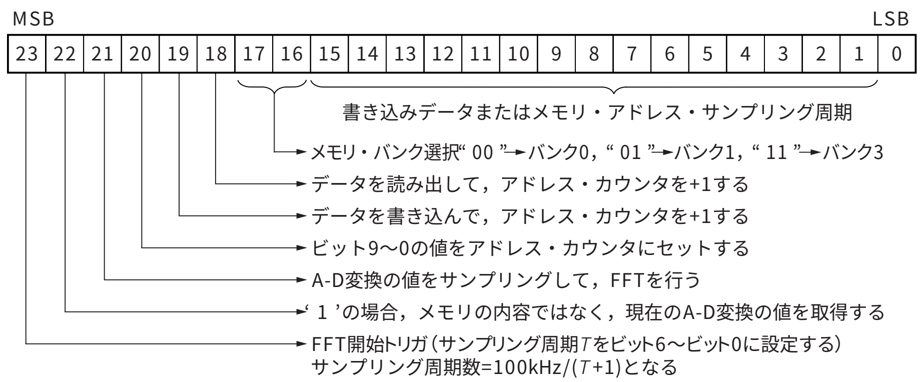


図11
コマンド・レジスタ

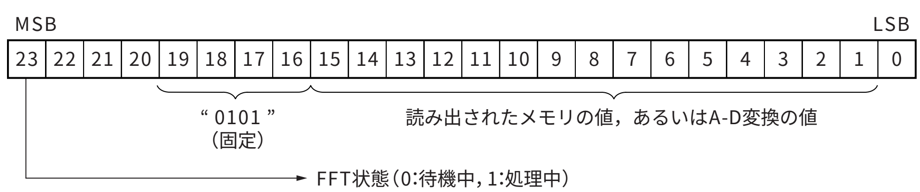


図12
ステータス・レジスタ

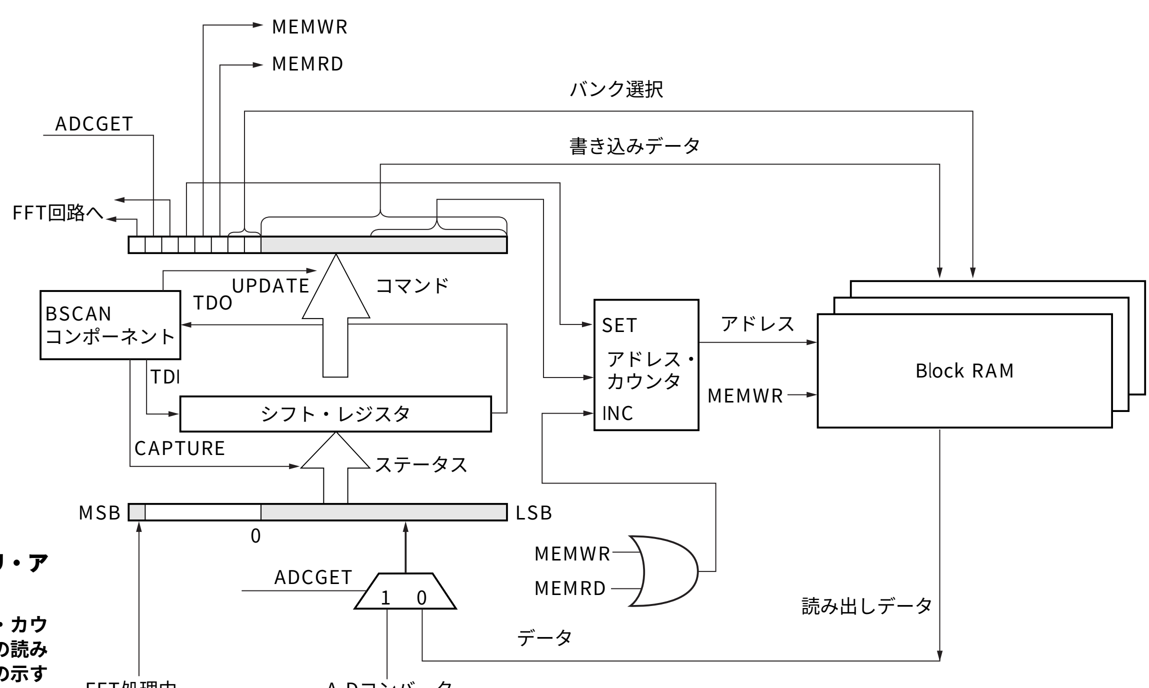


図13
JTAGによるメモリ・アクセス回路の構造
FPGA内にアドレス・カウンタがあり，メモリの読み書きはこのカウンタの示すアドレスに対して行われる．

リスト4
Block RAMの内容を読み出す

```
void read_mem(int bank,int addr,int len,fftint16 *data)
{
      unsigned char tdi[25];
      unsigned char tdocap[25];
      unsigned long indata;

      //USER1インストラクションの送信
      memset(tdi,1,ir_len); // すべて1にセット
      jtag_setbsreg_from_str(tdi,"000010",6);// USER1 (000010) をSpartan3にセット
      jtag_shift_ir(tdi,tdocap,ir_len);

      //データの送信
      indata = 0x00100000 | ((bank & 3) << 16) | (addr & 0x3ff); // set memory address
      jtag_setbsreg_from_ulong(tdi,indata,bsreg_len,0);
      jtag_shift_dr(tdi,tdocap,bsreg_len);

      //データの受信
      indata = 0x00040000; // memory read command
      jtag_setbsreg_from_ulong(tdi,indata,bsreg_len,0);
      for(int i=0;i<len;i++)
      {
              jtag_shift_dr(tdi,tdocap,bsreg_len);
              data[i] = jtag_getulong_from_bsreg(tdocap,16,0);
      }
}
```

● JTAGで通信するソフトウェア

本機は，米国Xilinx社のParallel Cable IVを介してFPGAと通信します．これらのJTAGケーブルはパラレル・ポートとJTAG信号線が1対1でつながっている構造なので，単純にパラレル・ポートに入出力するだけでアクセスできます．

Windows 2000/XPで動作するソフトウェアからI/Oアクセスを行うためには，何らかのデバイス・ドライバが必要になります．今回は，参考文献(6)の「PCIデバッグライブラリ for Win32」を使用します．このライブラリは，システム・ディレクトリまたはカレント・ディレクトリにPCIDEBUG.DLL，PCIDBG95.VXD，PCIDBGNT.SYSの三つのファイルがあれば，_IoReadCharまたは_IoWriteCharを呼ぶだけでI/Oアクセスが行えるものです．このライブラリはWebからダウンロードすることもできます(http://www.otto.to/~kasiwano/newpage16.htm)．

このライブラリのインストールは非常に簡単ですが，I/Oアクセスのたびにカーネル・モードへの切り替えが発生するので速度は遅くなります．そこでGIVEIOというモジュールを使ってアクセスする方法にも対応しました．GIVEIOがインストールされているパソコンでは，GIVEIOが優先的に使われます．GIVEIOはインストールはめんどうですが，カーネル・モードへの切り替えがない分だけ高速です．

本ソフトウェアを起動すると，JTAGケーブルの先に接続されているデバイスの構成を確認します．デバイスが確認されるとBlock RAMに，回転子と窓関数のデータを書き込み，ユーザからの操作を待ちます．

スペアナ・モードでは，FFTの後，512ワードの演算結果(複素数で256ポイント)の内容を読み出します．これは演算結果の半分の量です．一般に実数データをFFTすると半分の周波数のところで折り返した形のスペクトルになるため，半分だけ読めば十分です．

FFT計算モードでは，ホスト・パソコンでデータを計算してそれをFPGA内のメモリに転送し，FFTを行って表示します．

普通にサンプリングしたデータをそのままFFTするといろいろな問題が生じるので，窓関数という処理を施すのが一般的です．本機は窓関数用の係数メモリを持っていて，このメモリに任意の形の窓を書き込むことができます．本ソフトウェアは，ハニング窓，ハミング窓，方形窓などに対応しています．

5．製作した回路の動作と性能評価

FFTしたデータをパソコンに取り込むと，画面の更新は毎秒8回程度です．これは，JTAG通信の速度がボトルネックとなっています．

パラレル・ポートからTCKを操作するのに上げ下げで2ステップ，TDIの操作とTDOの入力も含めると4ステップを要します．パラレル・ポートのアクセス速度は，通常のPC/AT互換機なら約1MHzなので，データの転送速度は250kbps程度となります．本機では24ビットのレジスタに

データを入出力しますが，TAPの制御なども含めると7〜8回/sのデータの入出力が限度となります．

このようにプリンタ・ポートの速度がボトルネックになっています．しかし，XC3S50のJTAGクロックは33MHzまで上げることができるので，より高速にTCKを動かせる専用のハードウェアを作れば，最大で100倍程度の高速化が見込めます．

● Pentium 4と比較すると…

690 μsというFFTの速度が速いのかどうかということですが，同じ計算を2.4GHzのPentium 4で実行したところ，最短で70 μs程度でした．もちろん単純な比較はできませんが，CPUのほうがクロックが60倍速いのに対して性能は10倍にとどまりました．計算の効率はFPGAのほうが6倍良いといえます．

● Spartan-IIと比較すると…

さて，Spartan-IIで同じ回路を作った場合，どのくらいの性能になるのでしょうか？

XC2S50の場合，16×16ビット乗算器（符号なし整数）1個で全体の約20%の論理ブロックを消費します．XC3S50とXC2S50とではBlock RAMの構成が違いますし，乗算の際の符号を考慮していないので正しくは動作しませんが，無理やりフィットさせたところ，全体の約98％の論理ブロックを消費し，動作速度も39MHzとなりました．

このことからも，Spartan-3の乗算器がディジタル信号処理に有効であることがわかります．

6. まとめ

実は，このハードウェア・フーリエ変換機は，筆者が約8年前に考案し，標準論理ICとハードウェア乗算器，GALなど，合計60個程度のICを使ってISAバスの万能基板上に組み上げた複雑な回路でした（**写真2**）．それが1チップでできてしまうのですから時代の進歩を感じます．

このようにFFTを1チップ化することができましたが，FPGAのリソースはまだ44％しか消費していません．窓関数やA-Dコンバータのインターフェースなどを除いた，純粋にFFTの部分だけなら30％台です．

さらに余力のある人は，次のような改造を行ってみるとおもしろいでしょう．

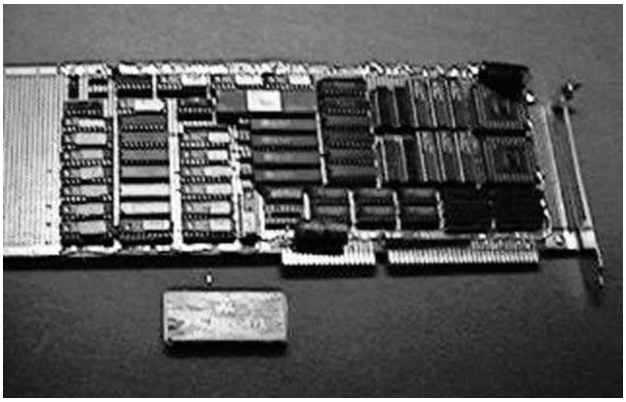


写真2　8年前に製作したハードウェア・フーリエ変換機
標準論理ICとハードウェア乗算器，GALなど，約60個のICを使用した．

- アンチエイリアシング・フィルタの実装
- Δ-Σ型A-Dコンバータの実装
- 逆FFTとΔ-Σ型D-Aコンバータの実装
- $\log\left(\sqrt{(x^2+y^2)}\right)$ の実装

スペクトラム・アナライザで遊ぶ際には，奇声を発して周囲の迷惑にならないように，くれぐれもご注意ください．

今回の設計データや関連ファイルは，本誌の付属CD-ROMに収録しています．また，ソフトウェアのアップデートやFPGAの回路の改良などは，筆者のホームページ（http://www.nahitech.com/nahitafu/）で公開していく予定です．

参考・引用*文献

(1) Spartan-3 デバイスでのエンベデッド乗算器の使用，アプリケーションノート，XAPP467（v1.1），Xilinx，2003年5月13日．
(2) Spartan-3 FPGAでのブロックRAMの使用，アプリケーションノート，XAPP463（v1.1.2），Xilinx，2003年7月23日．
(3) MCP3204データシート，http://ww1.microchip.com/downloads/en/DeviceDoc/21298b.pdf，Microchip Technology，1999年．
(4) LM324データシート，http://www.national.com/JPN/ds/LM/LM324.pdf，National Semiconductor，2000年8月．
(5) IEEE Standartd Test Access Port and Boundary-Scan Architecture，IEEE Std 1149.1-2001，IEEE．
(6) PCIデバイス設計入門，CQ出版，1999年12月．

ないとう・りゅうじ
(有)ナヒテック

<筆者プロフィール>
内藤竜治．技術士補（電気電子部門）資格を持つ．JTAGソフトウェアの開発を通じて経済産業省 独立行政法人情報処理推進機構によりスーパークリエータとして認定される．現在，新しくておもしろいJTAG応用技術の研究開発およびコンサルテーションを行っている．